Chapter ?

APPLYING GENERAL USABILITY SCENARIOS TO THE DESIGN OF THE SOFTWARE ARCHITECTURE OF A COLLABORATIVE WORKSPACE

Rob J. Adams, Len Bass, and Bonnie E. John
Carnegie Mellon University

Abstract: Architecturally-sensitive usability scenarios are important usability concerns that require early consideration in software design so that architectural support can render them easy and cost-effective to implement. Examples include providing the ability to cancel a command, undo commands, aggregate data, etc. This chapter reports on our experiences applying these scenarios to the design of MERBoard, a wall-sized interactive system developed by NASA to assist Mars Rover science teams with collaborative data analysis. We applied the scenarios during a major redesign of the software architecture that introduced usability as a valued quality attribute. In the process, we found that the scenarios were well-received by developers who readily understood how they related to MERBoard, that they applied to a collaborative workspace despite having been initially developed for a single-user desktop system, that they had a real impact on the architecture redesign, and that the scenario consideration process was quick and not too onerous for any of the team members.

Key words: software architecture, usability

1. INTRODUCTION

The usability analyses or user test data are in; the development team is poised to respond. The software had been carefully modularized so that modifications to the UI would be fast and easy. When the usability problems are presented, someone around the table exclaims, “Oh, no, we can’t change THAT!” The requested modification or feature reaches too far in to the

architecture of the system to allow economically viable and timely changes to be made. Even when the functionality is right, even when the UI is separated from that functionality, architectural decisions made early in development have precluded the implementation of a system with an acceptable level of usability. The members of the design and development teams are frustrated and disappointed that despite their best efforts, despite following current best practice, they must ship a product that is far less usable than they know it could be.

Over the past five years, our research group has worked to analyze the causes of the problem described above and to develop materials to help prevent it from occurring in common practice. This chapter describes these materials and relates our experiences applying them to the NASA MERBoard software development project. First, we review the relevant prior work on bringing usability concerns to software architecture design. Next, we describe the Usability and Software Architecture (U&SA) project's approach to the problem and provide an overview of the materials we have developed. We list the questions we had about our technique prior to our intervention with the MERBoard team, describe the procedure we went through during our intervention, and then conclude with our answers to our initial questions and an overview of our current ongoing work.

2. USABILITY AND SOFTWARE ARCHITECTURE

Historically, software engineers viewed usability as relevant to software architecture design solely through modifiability (Bass, Clements, Kazman, 1998, p. 78). If the user interface was sufficiently separate from the main application functionality, they argued, then the interface designers could make modifications through iterative design and testing throughout the project's life cycle, thereby maximizing usability. These engineers developed "separation patterns", or generalized architecture designs that separated the user interface into components that could change independently from the core application functionality. The Java 2 Platform, Enterprise Edition (J2EE) Model-View-Controller (MVC) pattern, shown in Figure 1, is an example of one of these (Sun Microsystems, Inc., 2003).

The separation patterns are highly successful at making "screen-deep" interface changes easy, for example, changing the size of the fonts to make them easier to read or the order of screens in a wizard to provide a more intuitive flow. Unfortunately, as our opening story illustrates, many important usability concerns are difficult to add late in the development process, even when the architecture is designed to follow one of the separation patterns. For example, often designers discover during testing

that users want to cancel long-running commands. To add this functionality to a MVC-based architecture, however, requires changing the View to add a cancel button, adding a Controller that runs on a separate thread (thus possibly introducing multi-threading in a single-threaded application) to listen for the cancel request, and modifying the command itself in the Model so it can cleanly cancel its execution and roll back to its initial state. As a result, the development team frequently finds that making commands cancelable is too expensive a change to make late in the development process. The software is released without this capability, and as a result is less usable than the team knew it could have been had they considered the cancellation requirement up front.

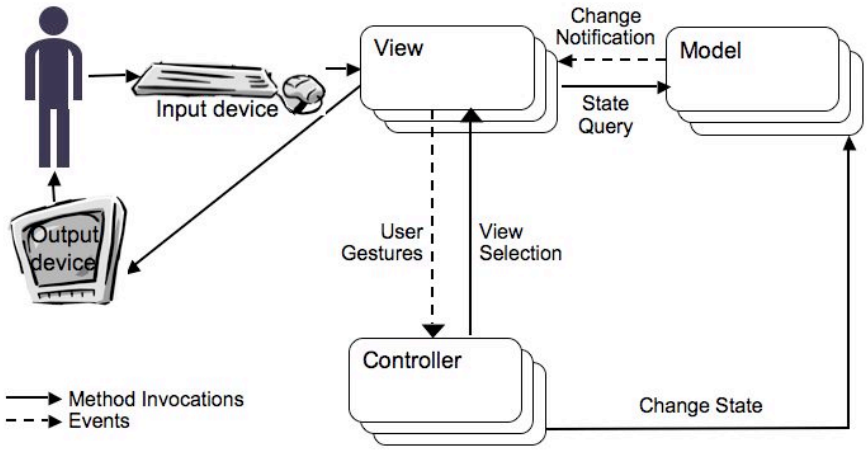

Figure 1: The J2EE Model-View-Controller software architectural separation pattern (Sun Microsystems, Inc., 2003). Arrows represent control flow, while boxes represent the major software components. The layered boxes indicate the existence of several instances of the component type.

3. THE USABILITY AND SOFTWARE ARCHITECTURE PROJECT

Since its inception, the Usability and Software Architecture (U&SA) project has worked to prevent the story that began this chapter. We envision a world in which routine practice brings important usability concerns to the table early enough that architectural limitations do not prevent them from getting implemented. To bring this about, we have the following goals:

1. Have usability recognized as a software quality attribute at architecture design time along with other quality attributes such as performance, maintainability, reliability, and security.
2. Understand and codify how usability impacts the architecture of software systems.
3. Improve communication between usability professionals[1] and software developers at the critical architecture design phase.
4. Provide guidance on designing architectures that support usability concerns.

For these goals to become a reality, we hypothesized that development teams required materials that clearly defined how to bring the knowledge and skills of the usability professionals and designers as well as the outputs of their design processes into the architecture design stage of the software development lifecycle. We developed the U&SA materials to satisfy this need.

3.1 U&SA Materials

In brief, our materials include a list of architecturally-sensitive usability scenarios, or generalized usability concerns that require difficult-to-change architectural support. Each scenario is connected to a hierarchy of usability benefits that break down *usability* into various components, such as *accelerating error-free portion of routine performance, preventing mistakes* and *supporting problem solving* which help give usability professionals a sense of what positive impacts implementing the scenario will have on the system's overall usability. The scenarios are decomposed into responsibilities of the software, which define the tasks the system must perform to properly implement the scenario as a list of requirements. For assistance with implementing the responsibilities, we provide architectural patterns that describe example implementation strategies within a particular architectural context. Finally, we describe the software engineering tactics that we employed in developing this implementation solution.

3.1.1 Architecturally-Sensitive Scenarios

At the time of this writing, we have identified 27 architecturally-sensitive usability scenarios. By *architecturally-sensitive*, we mean that support for

[1] In this paper we use the term "usability professionals" to include usability specialists, human factors specialists, ethnographers, interaction designers, graphic designers and other members of the project team who are primarily concerned with user-centered issues as opposed to primarily concerned with software architecture or detailed software design and implementation.

each scenario affects the functional core in a software architectural pattern based on separation of the UI, such as the J2EE-MVC. These scenarios are common to many interactive software systems and are not related to the domain functionality of any one system.

We generated scenarios by (1) reading several standard HCI textbooks and used their examples and definitions of usability to inspire scenarios (e.g. (Gram and Cockton, 1996; Newman and Lamming, 1995; Nielsen, 1993; Shneiderman, 1998)), (2) from our own experiences, and (3) through discussion with colleagues. Thus, the generation process was bottom-up, not theory-driven, systematic or comprehensive. However, it was sufficient to demonstrate that common usability concerns had implications for software architecture design.

The full list of scenarios can be found in Figure 9 (or see Bass & John, 2003 or Bass, John & Kates, 2001, for the scenarios themselves). A few examples are "Supporting Undo", "Canceling Commands", and "Reusing Information" (which will be our running example). Each scenario consists of a name and a paragraph or two describing the situation in which it occurs. The scenario for "Reusing Information" is shown in Figure 2.

Reusing Information

A user may wish to move data from one part of a system to another. For example, an administrative assistant may need to move a large list of business contacts from a word processor to a database. Re-entering this data by hand could be tedious and/or excessively time-consuming. Users should be provided with automatic (e.g., data propagation) or manual (e.g., cut and paste) data transports between different parts of a system. When such transports are available and easy to use, the user's ability to gain insight through multiple perspectives and/or analysis techniques will be enhanced.

Figure 2: The "Reusing Information" general scenario description.

The scenarios are intended to assist designers and usability professionals in identifying usability concerns that have architectural implications. Scenarios have a long history of applicability to user-interface design (Rosson & Carroll, 1992) and many designers and usability professionals are already familiar with them. In the spirit of Rosson & Carroll, our scenarios are "the things users characteristically want to do and need to do" (p. 183), but they are a lower level than the functionality-level of Rosson & Carroll's use scenarios, because usability issues show up at a lower level and architectural decisions must be made to support that level of use. Scenarios also appear in software development (albeit in different forms) in the Architecture Tradeoff Analysis Method[SM] (ATAM[SM], Kazman, Klein,

Clements, 2000) and in UML (Fowler, 2003) in the form of use cases, growth, and exploratory scenarios. Our scenarios are perhaps most similar to customer stories in Extreme Programming (Beck, 1999), as they are "one thing the customer wants the system to do" (p. 179) that is testable and can be implemented in one to five weeks. Thus, we hypothesized that they would serve as an effective cross-cultural communication device.

3.1.2 Usability Benefits Hierarchy

Each architecturally-sensitive usability scenario is allocated to the Usability Benefits Hierarchy, shown in Figure 3. The Usability Benefits Hierarchy describes the specific usability attributes of the system that implementing the scenario will enhance. Because there was no guarantee that architecturally-sensitive usability scenarios would span previous definitions of usability, we again used a bottom-up approach, affinity diagramming (or KJ-method, Kawakita, 1982), to organize the scenarios into topics. Although it is not directly derived from other published definitions of usability, the Benefits Hierarchy covers the same general concepts of efficiency, error prevention and tolerance, and user satisfaction, as other popular usability definitions (e.g., ISO 9241-11:1998; Newman and Lamming, 1995; Nielsen, 1993; Shneiderman, 1998). It does not cover user satisfaction in any depth, however, neglecting concepts like physical discomfort, for example (ISO 9241-11:1998). However, it includes benefits relating to reducing the impact of system errors that other usability definitions do not include.

Increases individual user effectiveness

Expedites routine performance

Accelerates error-free portion of routine performance

Reduces the impact of routine user errors (slips)[2]

Improves non-routine performance

Supports problem-solving

Facilitates learning

Reduces the impact of user errors caused by lack of knowledge (mistakes)

Prevents mistakes

Accommodates mistakes

Reduces the impact of system errors

Prevents system errors

Tolerates system errors

Increases user confidence and comfort

Figure 3: The Usability Benefits Hierarchy. For each scenario, the U&SA technique describes which specific benefits (the "leaves" of the hierarchy) apply and which do not. An example is shown in Figure 4.

For each scenario, the U&SA technique describes which specific benefits (the "leaves" of the hierarchy) apply and which do not. Figure 4 contains the allocation of the "Reusing Information" scenario to the Benefits Hierarchy. For each benefit allocation, we include a short justification for why the benefit applies to this scenario.

3.1.3 Responsibilities[3]

Each scenario package includes a list of system responsibilities that can serve as a specification to developers, detailing what the system must do.[4] Like any specification, the responsibilities are intended to describe the functions of the system without dictating a particular implementation. The responsibilities for "Reusing Information" are divided into two sections: manual reuse (i.e., copy&paste) and automatic reuse (data propagation). These responsibilities are shown in Figure 5.

---

[2] The distinctions between *errors*, *slips* and *mistakes* in the Usability Benefit Hierarchy follow Norman, 1983.

[3] *Responsibility* is a term from object-oriented design that means "an obligation to perform a task or know information" (Wirfs-Brock & Mckean, 2003, p. 3).

[4] At the time of the intervention with the NASA development team, only 6 scenarios out of 27 included this list of responsibilities. Work continues to fill these in for every scenario.

Increases individual effectiveness
Expedites routine performance
Accelerates error-free portion of routine performance

In most cases, it is more efficient for systems to transport information from place to place than it is for users to re-enter this information by hand. Thus, systems that support information reuse accelerate routine performance.

Increases individual effectiveness
Expedites routine performance
Reduces impact of slips

Automatic data transportation and/or re-entry require fewer human actions (e.g., typing, mouse movements) than re-entering data by hand. Since performing more actions introduces more opportunities for error, systems that support information reuse can prevent slips.

Increases individual effectiveness
Improves non-routine performance
Supports problem-solving

When users can import and export data from one place to another easily, they may try different applications to gain additional insight while solving problems. For example, a user may export data from a traditional text–based statistics application to a data visualization application. Thus, systems that support information reuse facilitate problem-solving.

Figure 4. Allocation of "Reusing Information" to the Usability Benefits Hierarchy. A system that supports reusing information impacts the benefits listed above, but has little or no influence on the other benefits in Figure 3.

Manual Reuse Responsibilities

R1. Provide information to be reused (from Information Source)
R2. Store information to be reused (in Information Repository)
R3. Provide feedback on the stored information
R4. Retrieve stored information (from Information Repository)
R5. Receive information (into Information Sink)
R6. Provide feedback on the retrieved information

Automatic Reuse Responsibilities

R1. Know which data to store and retrieve from repository (e.g., via a data dictionary)
R2. Provide information to be reused (from Information Source)
R3. Store information to be reused (in Information Repository)
    (a) Retrieve stored information on request
    or
    (b) Broadcast newly stored information
R4. Receive information (into Information Sink)

Figure 5: Responsibilities for Reusing Information

3.1.4 Architectural Patterns

To provide more guidance to software developers, we have included a sample architectural pattern in each U&SA scenario package that fulfills the implementation-independent responsibilities. These patterns are similar to software patterns (Gamma, Helm, Johnson and Vlissides, 1995) insofar as they describe generalized solutions that could be realized in a wide variety of systems, but most are at a level of abstraction similar to software architecture patterns (Buschmann, Meunier, Rohnert and Sommerlad, 1996).

Because the architectural patterns that support usability are always situated within an overarching architecture (usually a separation-based architecture discussed above), our examples must be given with respect to some overarching architecture. We have chosen to situate our examples within the J2EE Model-View-Controller architecture because that pattern is very popular in modern system development (Figure 6). However, the concepts illustrated in each example can be applied to other overarching architectures.

Note that the pattern defines generic, high-level components and the interactions between them. Each responsibility, listed in the previous section, is allocated to a particular component, as described in Figure 7.

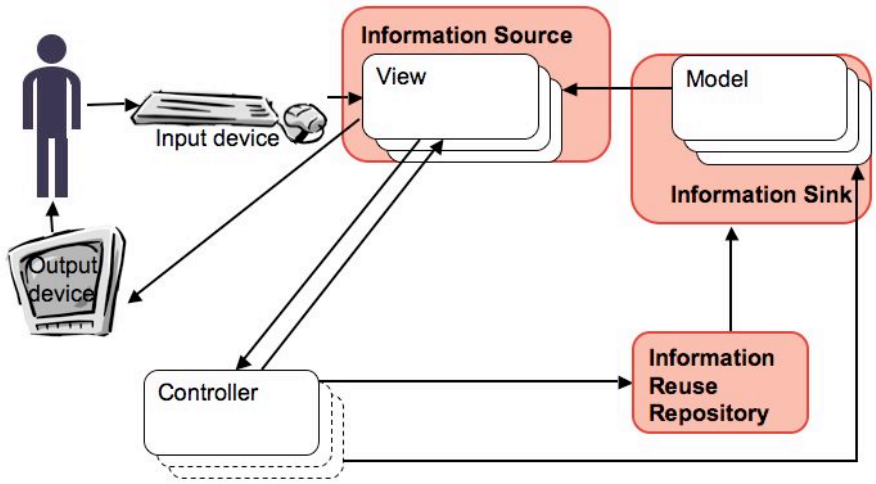

Figure 6: Sample architectural pattern for *Reusing Information Manually*.

Allocation of Responsibilities for Reusing Information Manually

View

- Accept copy/paste commands from the user (R1)
- Send data to the Controller (R1)
- Provide feedback about the copied data. (R3)
- Provide feedback about the pasted data. (R6)

Controller

- Send data to the Information Reuse Repository (R1)
- Send information about the copy operation to the View. (R3)

Model

- Receive data from the Information Reuse Repository (R5)

Information Reuse Repository (which is a Model)

- Receives data to be reused, e.g., from the Controller in response to a copy request (R2)
- Stores information to be reused (R2)
- Accepts commands to retrieve stored information, e.g., paste to the Model (R4)
- Dispense information to be reused to requesting Models. (R4)
- Provide information to the View for user feedback about the repository contents. (R3)

Figure 7: Allocation of Responsibilities for *Reusing Information Manually*. This figure describes the mappings between the Reusing Information Manually responsibilities (in parentheses) and the components shown in the sample architectural pattern in Figure 6. Some responsibilities require participation by more than one component as indicated by their labels appearing in multiple components.

3.1.5 Software Tactics

The last part of a U&SA scenario package includes a list of the architectural tactics employed by the sample architectural pattern to implement the scenario. These architectural tactics, design decisions that influence quality attributes like usability or performance, were developed to codify best-practice solution techniques for common software design problems (Bass, Clements, & Kazman, 2003). The software tactics hierarchy for usability appears in Figure 8.

In the case of *Reusing Information Manually*, the sample architectural pattern uses the *data intermediary* tactic to implement the information reuse repository component. Most of the architecture examples for the other usability scenarios employ multiple tactics to implement a solution.

Software Architecture Tactics Hierarchy

Localize modifications
- Hide information
- Separate data from commands
- Separate data from the view of that data
- Separate authoring from execution

Maintain multiple copies
- Data
- Commands

Use an intermediary
- Data
- Function

Recording

Preemptive scheduling policy

Support system initiative
- Task model
- User model
- System model

Figure 8: Software Architecture Tactics Hierarchy. For each scenario, we list the tactics used in our sample solution pattern.

3.1.6 Benefits / Tactics Matrix

In addition to our list of scenario packages, we developed a tool to help apply the U&SA materials to a development effort: the Benefits / Tactics Matrix, shown in Figure 9.

When a project team wishes to determine which scenarios are important for their system, they first assess which usability benefits are critical for fulfilling their usability goals. Then they read down the column of each benefit and find the scenarios they must consider during the architecture design phase.

After the development team has determined that their architecture design includes support for all the usability scenarios they have deemed critical, or if an architecture is already in place, the team may use the matrix to identify additional scenarios that may be easy to support. They enter the Benefits / Tactics Matrix through the software engineering tactics they have already employed and read across the rows to identify which scenarios may be easy for them to support with their existing design. Even though these scenarios are not critical, the team may wish to consider implementing them if the architecture they have chosen will support them without much additional effort.

Architectural Tactics ↓ / Usability Benefits →		Increases individual effectiveness						Reduces impact of system errors		Increases confidence and comfort
		Expedites routine performance		Improves non-routine performance		Reduces impact of mistakes				
		Accelerates error-free portion	Reduces impact of slips	Supports problem-solving	Facilitates learning	Prevents mistakes	Accommo-dates mistakes	Tolerates system errors	Prevents system errors	
Localize Modifications	Hide information	4, 13, 14, 15, 20, 23		4, 13, 20	4, 13, 20	4, 13, 20	9, 14		23	
	Separate data from the view of that data	12, 13, 24, 25	12	12, 13, 22, 24, 25, 26	12, 13, 24	12, 13, 22, 24	12			12
	Separate data from commands	1, 24, 25	5, 17	5, 17, 24, 25, 26	5, 17, 24	1, 5, 17, 24	1, 5, 17			17
	Separate authoring from execution	1, 2	2			1, 2	1, 2			
Maintain multiple copies	Data	16								
	Commands	2	2	22		2, 22	2			
Use an intermediary	Data	7, 11, 14	11	7, 11			14			
	Function	6, 14, 20, 27	27	6, 20	20	20, 27	14		6	27
Recording		2, 7	2, 3, 21	3, 7, 21		2	2, 3, 21	3, 8		
Preemptive scheduling policy		15, 18, 19	3, 5, 17, 18	3, 5, 10, 17	5, 10, 17	5, 17, 19	3, 5, 17	3		17, 18
Support system initiative	Task model	18, 19	5, 17, 18	5, 10, 17	5, 10, 17	5, 17, 19	5, 17			17, 18
	User model	12, 18	5, 12, 17, 18	5, 10, 12, 17, 22	5, 10, 12, 17	5, 12, 17, 22	5, 12, 17			12, 17, 18
	System model	4, 6, 19, 23	3, 5, 17	3, 4, 5, 6, 17	4, 5, 17	4, 5, 17, 19	3, 5, 17	3	6, 23	17

KEY

1 Aggregating data
2 Aggregating commands
3 Canceling commands
4 Using applications concurrently
5 Checking for correctness
6 Maintaining device independence
7 Evaluating the system
8 Recovering from failure
9 Retrieving forgotten passwords
10 Providing good help
11 Reusing information
12 Supporting international use
13 Leveraging human knowledge
14 Modifying interfaces
15 Supporting multiple activity
16 Navigating within a single view
17 Observing system state
18 Working at the user's pace
19 Predicting task duration
20 Supporting comprehensive searching
21 Supporting Undo
22 Working in an unfamiliar context
23 Verifying resources
24 Operating consistently across views
25 Making views accessible
26 Supporting visualization

Figure 9: The Benefits / Tactics Matrix. The usability benefits are listed across the top of the table, the architectural tactics are listed down the side. The numbers in the cells refer to the specific scenario packages that give the column's benefit and employ the row's tactic. An additional scenario, Supporting Personalization, was added after this matrix was created.

3.2 Prior Uses of U&SA Materials

The U&SA materials described above had been developed and disseminated over the course of more than five years. Since we began work on this project in 1999, we have run several industry-focused tutorials on applying our materials (Bass, John, Juristo, & Sanchez-Segura, 2004; John, Bass, Juristo, & Sanchez-Segura, 2004; John & Bass, 2002, 2003), presented

our work at usability and software engineering conferences,[5] and published information on the U&SA materials in Software Engineering Institute technical reports and software engineering magazines. We have also applied the information in the scenario packages informally in a few architecture design reviews. For example, we used a few of the scenarios as part of the ATAM[SM] on a large commercial information system (Bass & John, 2003). However, the full set of scenarios had never been explicitly applied to a real-world software system undergoing a major architectural redesign. Therefore, although our materials appear useful, we still needed to subject them to the test of real-world use.

4. QUESTIONS FOR A REAL-WORLD CASE

We set out to test our materials by using them as the main discussion points for an architectural review of a real-world software project with significant architectural design problems and an emphasis on usability. Although we recognized that no single case could give us definite, generalizable answers to all our questions, we hoped to get feedback, suggestions, and new ideas that would help us refine our materials in preparation for more rigorous empirical studies. We set out with three specific questions, detailed below.

Would a real-world software development team accept the U&SA materials as the main discussion point of an architecture design meeting?

Traditionally, development teams have not considered usability as a software quality attribute at the architecture design phase. Usability issues are introduced much later in the life cycle through user testing and design iteration and earlier in the life cycle through ethnography, contextual inquiry, and other field techniques. Our experience has been that usability professionals are frequently not invited to architecture design meetings, and when they are, they feel they have little to contribute because they have no training in software architecture design or its implications for producing usable systems. We created the U&SA materials to a framework within which usability professionals could contribute to a software architecture design meeting.

We had successfully introduced our scenarios for enhancing usability as a quality attribute alongside more traditional architectural quality attributes such as performance, security, and reliability during broad architectural

[5] For a full list of references, see http://www.uandsa.org

reviews. However, as of mid-2002, usability had never been the main topic of discussion in a large-scale, real-world architecture design meeting. We were interested in discovering whether a development team confronting a larger software architecture design effort would accept usability as an architectural quality attribute and whether both the developers and usability professionals on the team would be able to use our scenarios to participate in a discussion about the system's proposed architectural design.

Would usability scenarios generated by considering single-user-at-a-desktop apply to a real-world design problem that may involve other domains (such as collaborative workspaces, web-based environments, etc)?

The U&SA scenarios were initially developed through literature investigations and examinations of usability problems in common desktop applications and operating system interfaces. Most of these "single-user-at-a-desktop" applications followed the classic WIMP paradigm, executed on a single machine only, and did not support multiple-user collaboration. Single-user-at-a-desktop does not cover all possible environments that have potential software architecture and usability issues, however. Modern systems are designed to support domains with requirements that span a wide variety of paradigms, including collaborative computer-supported cooperative work environments, real-time embedded systems, ubiquitous computing, and so on. We hoped to discover how many of our scenarios would apply in these other environments, which are different in many respects from the one we had in mind while developing the scenarios. Although no single case can cover all these environments, applying our materials to a system in any environment off the desktop is a step toward answering this question.

Would our architecture design suggestions contribute to a real design project?

Ultimately, the U&SA materials are designed to improve architectural decisions made early in the life cycle with respect to their support for usability. Thus, the purpose of the scenarios is to generate design suggestions for software architectures which, when followed, help to prevent the "We can't change *THAT*!" problem described in the introduction. In applying our materials to a real-world development project, we wanted to discover whether the scenarios could, in fact, suggest design changes to the proposed architecture of a real software system so we could learn whether our materials were effective at all. We also hoped to discover whether real

development teams would find these suggestions compelling enough to change their architecture design.

With these questions in mind we began to collaborate with the development team of the MERBoard project, a software development project at NASA Ames Research Center that is a participant in the High Dependability Computing Program[6]. As a participant in the HDCP, the MERBoard development team agreed to allow intervention by software engineering researchers for the purpose of testing new methods and tools.

5. THE MERBOARD PROJECT

The MERBoard Project is a software development effort by NASA Ames Research Center[7] to create a collaborative tool to support the engineers and scientists on the Mars Exploration Rovers (MER) mission.[8]

Two robotic probes landed on Mars in January 2004. The MER mission's scientific goals include searching for and characterizing a wide range of rocks and soils that hold clues to past water activity on Mars. The MER collects soil samples and other geological data from the Martian surface and transmits this information to NASA scientists back on Earth for analysis. Each MER is solar powered; during the Martian day, it collects data based on instructions sent to it from Earth. When night comes, it transmits this data back to Earth and goes into a low-power, low-activity mode until the sun rises in the morning. During the Martian night, scientists back on Earth must analyze the data received from the MER to determine what instructions to send to the robot in the morning. For instance, if the data indicate that there is a high probability that an old water channel might lie to the left, scientists must send orders to the MER to investigate that area in the morning. The scientists must be able to analyze the data and make decisions under strict deadlines, so that the MER does not sit idle.

To facilitate communication, the scientists work in a collocated, "war-room" style environment. Their initial technology support consisted of desktop and laptop computers running a variety of software applications, projection screens, and paper flip charts to facilitate group thinking and discussion. The MERBoard Project introduced new technology to support collaborative activities like annotating images and strategic planning with

[6] For information about the HDCP, see http://www.cebase.org/HDCP/frames.html?/HDCP/aboutus.htm

[7] For information about MERBoard, see http://ic.arc.nasa.gov/story.php?sid=104

[8] For information about the MER mission, see http://marsrovers.jpl.nasa.gov/home/index.html).

storage, retrieval and sharing capabilities (Tollinger, McCurdy, Vera & Tollinger, 2004).

The MERBoard is a wall-sized collaborative workspace intended to facilitate shoulder-to-shoulder collaboration (Figure 10). The physical hardware consists of a large touch-sensitive plasma display. The software consists of four major components: a web browser for on-the-fly internet research, a collaborative whiteboard for creating and annotating visualizations of data, a remote login (VNC) client for connecting the MERBoard to the scientists' desktop and laptop computers, and MERSpace, a shared document repository for saved MERBoard sessions.

Usability had always been a key goal for the MERBoard project; their slogan was that the final system had to be to be "Palm Pilot simple". The MERBoards are intended to enhance the productivity of the scientists, who have a wide variance in their comfort with new technology, are too busy to spend much time becoming familiar with the tool before the mission, and have tight deadlines during the mission. Thus, the system must be both easy to learn and efficient to use, two key aspects of usability.

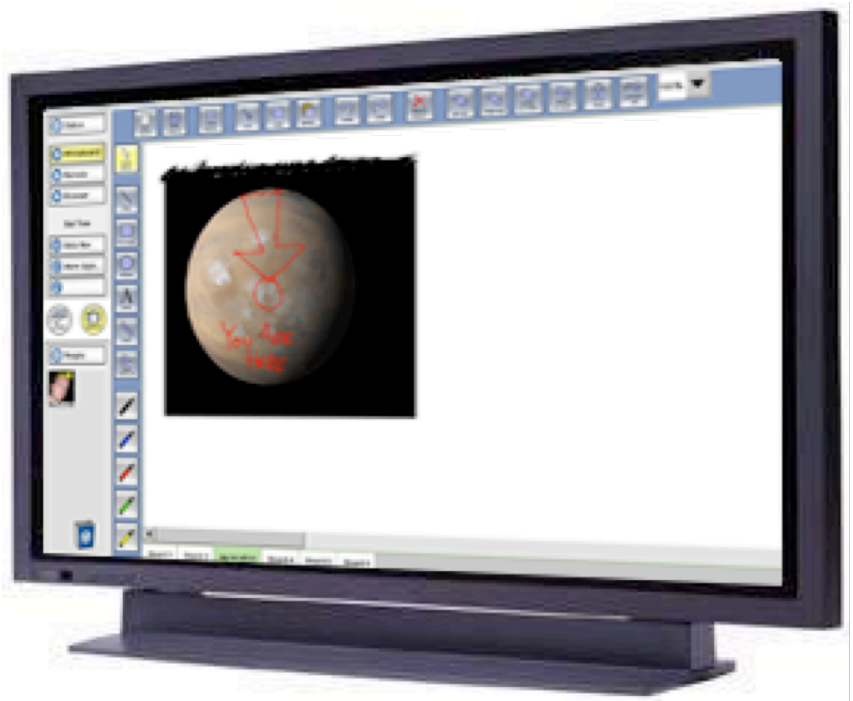

Figure 10: A photograph of the MERBoard's whiteboard screen (MERBoard User's Guide, NASA Ames Research Center, September 10, 2003, by permission).

5.1 MERBoard Project Timeline

The MERBoard team has operated in several phases with defined deliverables (Figure 11). For the first phase, beginning in Fall 2001, the MERBoard project team conducted ethnographic field studies and user research to determine the real needs of the engineers and scientists. They then began development on a working prototype that could be user tested in the 2002 summer field tests with other MER technology. After the tests were completed, the team took the issues identified in the usability tests and began a ground-up rewrite effort, this time with an emphasis on sound architectural design for extensibility, performance, and reliability. They began with an architecture redesign meeting to set their goals for the January 2004 landing of this MER mission and for the 2009 MER mission as well. Our intervention began at the September 2004 architecture review meeting and continued through teleconferences with a MERBoard developer.

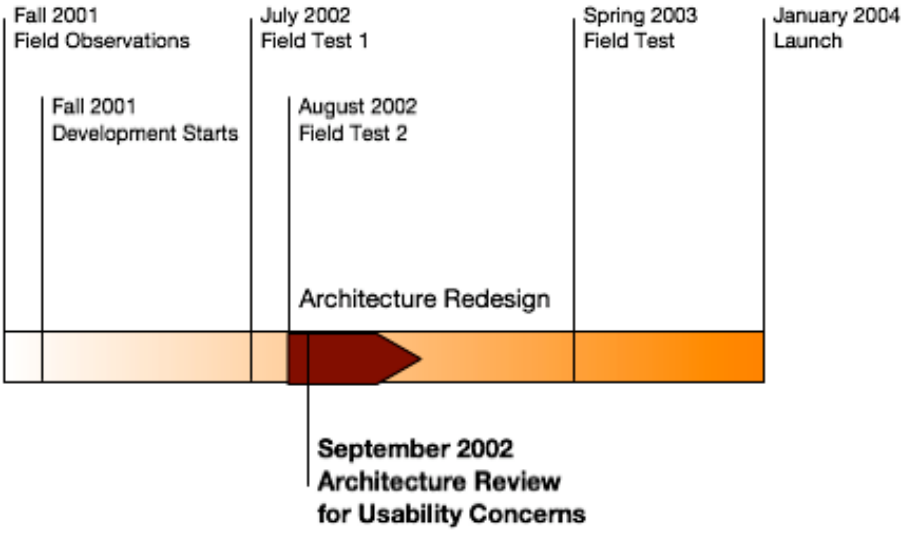

Figure 11: The MERBoard development timeline. We applied the U&SA materials during the architecture redesign phase at the September 2002 architecture review for usability concerns and follow-on teleconferences.

6. U&SA'S APPLICATION TO MERBOARD

Since MERBoard had articulated usability as a primary goal of their system from the beginning, we proposed that it be considered as an architectural quality attribute along with their other stated attributes of

extendibility, performance, and reliability. Since classic architecture design and analysis techniques do not address usability as a quality attribute, we offered to help the MERBoard team apply our U&SA materials to their proposed architecture redesign.

Our intervention took place over the course of four meetings: a face-to-face meeting where the lead architect walked through an overview of the proposed architecture redesign, a face-to-face meeting to introduce the MERBoard team to the U&SA materials and prioritize their usability goals, a teleconference with the front-end developer to review his understanding of the U&SA scenario packages, and a second teleconference with the front-end developer to review his application of the scenario packages to the details of his proposed architecture design.

6.1 Face-to-face meetings: Architecture overview and U&SA materials

The first meeting was an architecture overview for the MERBoard project team. It took place in the MERBoard project lab and involved the entire MERBoard project team, including the project manager, the usability professionals (including an ethnographer, several cognitive modelers, HCI specialists, and a graphic designer), the lead architect and several software developers. The second and third authors were primarily observers at this architecture overview, although we were invited to ask clarifying questions. The lead architect of the MERBoard system presented the proposed architecture redesign and discussed technical concerns such as what library to use to handle gesture input, how to structure the components to support future extensibility, etc. The project manager and software developers asked questions; the second author asked a few clarifying questions; the usability professionals were generally silent listeners. The meeting took approximately four hours.

There was then a break for dinner and the majority of the MERBoard team returned to hear us describe the U&SA materials and to prioritize the scenarios for the MERBoard release (one designer had a previous commitment and could not return). We gave a short overview of the U&SA motivation and approach, and then presented our list of scenarios that form the core of our scenario packages. We led the team through a review of their architectural requirements by going over each of our twenty-seven scenarios in turn. For each scenario, the team decided whether:

- The scenario applied to the current, January 2004 target (i.e., it must be supported by the redesigned architecture and implemented in the current release).

- The scenario applied, but they did not anticipate needing it until the distant 2009 release (i.e., it was safe to delay).
- The scenario did not apply to MERBoard.

In this meeting everyone, including the usability professionals, contributed to the discussion. Unlike the previous architecture overview, the entire team debated the needs of their users and what impact this would have on their architectural requirements.

By the end of the meeting, the design and development team had found that 25 of the 27 scenarios were applicable to MERBoard. Seventeen of these scenarios were considered essential for the January 2004 release and were targeted for the next field trial. Eight were determined less critical and were postponed for the longer-term release.

Since 93% our scenarios were judged applicable by the development team, we conclude that they were highly relevant to MERBoard, a real-world project with a significant architecture design challenge. Moreover, the team accepted our scenarios as a means of discussing usability as a software quality attribute that applied to their system's architecture. Even more encouraging was the nature of the discussion our technique fostered in the team; the usability experts and software experts had common ground on which to discuss critical design decisions at a sufficiently early stage for changes to be made.

6.2 Teleconference to review U&SA materials

At the initial face-to-face meetings, the MERBoard management determined that most of the relevant U&SA scenarios applied to the design of the front-end of MERBoard, as opposed to the back-end (or server-side). Therefore, we arranged follow-up discussions with the front-end architect and developer (hereafter, FED). It was arranged that these discussions would be via teleconference because the authors and the MERboard team were separated by 3000 miles and travel budget for both groups was limited. We provided FED with a copy of our technical report on the U&SA scenarios (Bass, John, and Kates, 2001) as well as the notes packet to our 2002 CHI tutorial on applying the U&SA technique (John and Bass, 2002). FED read these materials during a four-day period that spanned a weekend, while he redesigned the front-end architecture.

The following week, we had a teleconference with FED to get his reaction to the scenario packages and our technical report. There were four participants in this teleconference: FED (at NASA Ames in California), architecture expert Len Bass, usability expert Bonnie John, and research associate Rob Adams (at Carnegie Mellon University in Pittsburgh, Pennsylvania). We solicited the FED's opinions on the patterns, whether

and how he felt they applied to MERBoard's architecture design, and clarified those issues about which he was uncertain. We discussed his general impressions of the U&SA materials as a whole, and then went through each scenario package in order to get his specific impressions on those that the team had decided were critical for the current release. FED described to us how he foresaw each scenario package influencing the technical decisions he was facing. The entire discussion lasted approximately one hour.

FED's reactions to the U&SA materials were primarily positive. Referring to the U&SA scenario packages as a whole (i.e. the scenarios, usability benefits, architecture patterns and software engineering tactics), he said

> "It's nice to explicitly describe it like this. I mean I managed to avoid any actual classes that actually taught architecture, this kind of design patterns, you know software engineering. So this is basically how I would write... I think I'd write [the architecture like this] anyway but it's definitely is nice to have it laid out and drawn up and written up for you. And then you can say okay this is how we're going to do it. As opposed to here's my, sort of, thoughts on the matter."

> "…it's also nice just keeping a list [of scenario packages] next to me so when I'm doing my design decision I can glance at it to make sure, you know, I haven't forgotten anything."

About the architecture patterns associated with each scenario, FED said they were "very clear" even though he did not have experience in software patterns or architecture patterns prior to using the U&SA materials. About applying them to the MERBoard front-end architecture redesign he said,

> "So, they're pretty interesting…Of the ones that tools actually used, the patterns, some patterns were somewhat useful others weren't... [some patterns] didn't really apply. And I guess some were sort of already there.., [the pattern in the U&SA documents] described something that already exists [in the MERBoard architecture]. So it's not actually wrong, it's confirmation that we're doing something right."

For example, regarding the Aggregating Command Scenario, FED judged the proposed architecture for the MERBoard's whiteboard as "very very similar to this pattern … the grouping manager and command cluster … have this separation described in the pattern."

Unsolicited by the researchers, FED mentioned that having a separate list of responsibilities fulfilled by the pattern was helpful (such a list was available for six patterns at the time of this intervention, in John & Bass, 2002).

"…the breakdown of responsibilities was quite nice, I felt. It wasn't critical but it definitely made it a lot easier to think about."

On a less positive note, when speaking about the software engineering tactics, FED was polite, as would be expected in such a discussion with researchers who developed the materials under discussion. He said they were "probably definitely helpful", but could not think of any concrete instances of how these tactics were useful to him. He thought they would be more useful if they were integrated into the description of the example architecture patterns as "key ideas" used in each pattern.

In summary, FED expressed that he was able to understand the U&SA materials and connect them to the MERBoard front-end architecture he was designing. We arranged to have an additional teleconference once he had documented his architecture design and review that design with respect to the scenarios.

6.3 Teleconference to specifically apply U&SA materials

In advance of our second teleconference (with the same participants), FED sent us a diagram of his proposed architecture design. We went through all the scenario packages that the design and development team had deemed necessary for the 2004 release and discussed how the proposed architecture supported each scenario package. The architecture expert and FED each proposed changes to the diagramed architecture in light of the considerations raised by the scenario packages, then discussed and decided on those changes. This meeting ran for approximately one hour.

6.3.1 General impressions of the application of U&SA materials

The discussion in this teleconference was a collaboration between FED, who was an expert on MERBoard but had no formal training in software architecture (as had been uncovered during the first teleconference), and the U&SA researchers, primarily the software architecture expert. The conversation reflects this collaboration in that 46% of the words were uttered by FED, indicating that it was not a "lecture" by the architecture expert, who uttered 44% of the words. Had it been a lecture by the architecture expert, a larger percentage of the words would have been uttered by that researcher. Nor was it a "seeded" design review where the architecture expert throws out an idea and the domain expert then dominates, or a larger percentage of the words would have been uttered by FED. Since the development team had already decided which scenarios were important to the MERBoard, the usability experts' input to this discussion was small (5% of the words), primarily asking clarifying questions in order to take notes and revise the

architecture diagram. The more junior research associate primarily asked clarifying questions (5% of the words).

In the previous teleconference, FED expressed confidence in his understanding of the U&SA materials and in this teleconference he seemed readily able to apply the general scenarios to his specific architecture design problem; each scenario immediately brought to mind a specific technical challenge he was facing and he was able to use these scenarios to brainstorm potential implementation solutions. However, FED seemed less able to apply the component-level patterns we provided in the technical report to MERBoard without additional support from the U&SA team, as evidenced by the large number of changes we made during this review, described below. In one respect this shows that U&SA materials and expertise can have a influence on architecture design. On the other hand, this is evidence that the U&SA materials need to be improved for them to become a stand-alone resource for software architects in the real world.

Moving from general impressions to specific content of the teleconference, the next section details the proposed MERBoard front-end architecture and the changes we made during this teleconference.

6.3.2 Results of the U&SA Intervention on the MERBoard Architecture

The architecture diagram FED sent us at the beginning of the second teleconference is shown in Figure 12. The architecture that resulted from the discussions during that teleconference is shown in Figure 13. The components in Figure 13 and their responsibilities are as follows.

- The **GUI** contains all the user interface widgets that appear on the MERBoard and handles user input processing logic. The GUI is implemented using the Java Swing user interface toolkit.
- The **Dispatcher** receives user actions from the GUI and either handles them itself or forwards them to the appropriate component for processing.
- The **Administrator** handles all user management and personalization functions.
- The **Selector** provides a number of utilities relating to the display and manipulation of user and personalization information, thereby acting as a bridge between the user interface and the Administrator.
- The **Save / Restore Interface** takes snapshots of the MERBoard's current state and sends them to the server over the network. This allows the MERBoard to be restored in case of a system crash, minimizing data loss. It also handles manual requests for saving and restoring data.

- The **Recorder** logs usage data for later analysis by the usability professionals to identify usability breakdowns and areas that need improvement. These data are intended to feed into future collaborative systems developed by NASA.
- The **Network Interface** provides an abstraction layer for communication with the remote server component on which the MERBoard's data is saved. The remote server is not shown on the diagram.
- The **Plugins** implement specific functionality extensions to the MERBoard. The plugins developed by the MERBoard team include the whiteboard, the web browser, the VNC-based plugin for connection to a remote computer, and a specialized tool for the Long-Term Planning group called the "Sol Tree Tool".

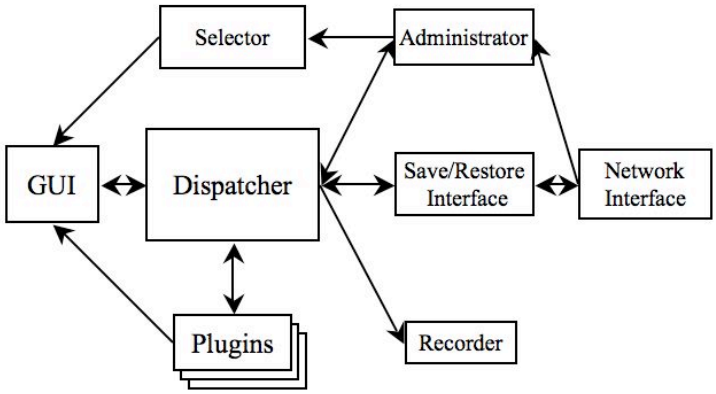

Figure 12: MERBoard architecture diagram proposed by FED prior to the second teleconference. The developer created this diagram after being exposed to the U&SA materials, but before consulting with U&SA researchers in detail about each scenario.

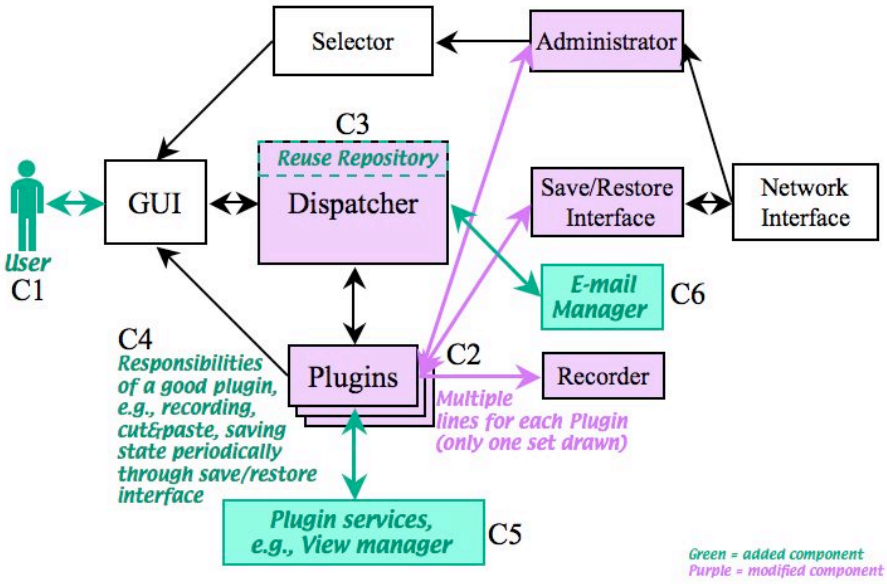

Figure 13: The modified MERBoard architecture diagram, developed collaboratively by the FED and U&SA researchers during the teleconference where the proposed architecture was discussed with reference to the U&SA scenarios relevant to the January 2004 release. The changes made are labeled C1 through C6 (these labels do not appear in the architecture diagram used by the developers).

Comparing Figures 12 and 13, it is easy to see that almost every component and communication line was either modified or added because of the detailed discussion of the U&SA scenarios. We audiotaped the teleconference, which allowed us to identify when these changes were made in the discussion and determine what information content led to each change. Below, we examine each change by considering whether the U&SA scenario packages directly caused the change or if other aspects of this architecture walkthrough steered the design. Figure 13 labels each change, C1 through C6.

The first modification (C1) involved the addition of a representation of the MERBoard user to the diagram, thus giving a sense of where the user fits into the system. This constituted a simple omission on FED's part, a common occurrence when documenting complex systems from a software engineering point of view. The usability expert suggested the addition to keep the user evident in the architecture documentation. It is possible that any review of the diagram by an independent person taking a human-

centered approach could have turned up this omission; no special U&SA scenario package can be credited with this addition.

The next modification (C2) involved altering the communication paths between the Plugins, the Dispatcher, and the other components. The intent of this modification was to simplify the communication between the Plugin and the worker components (the Administrator, Save / Restore Interface, and Recorder) so that these potentially heavy communication channels would not all have to be routed through the Dispatcher. This change arose from a general discussion of the architecture. The architecture expert suggested this change to improve the overall conceptual integrity of the MERBoard design (a quality attribute he called "buildability"). There was no explicit reference to any U&SA scenario package.

The addition of the "Reuse Repository" in the Dispatcher (C3) addresses the need for an explicit sink for copied and pasted data (commonly known as a *clipboard*) and also speaks to the need for defined mechanisms for handling and transporting clipboard data between components. This addition arose as the result of a long discussion of U&SA's Reusing Information scenario package. The front-end developer explained his implementation of information reuse in the MERBoard and the merits of various alternatives with the architecture expert. Unlike the previous two examples, this change arose directly from the discussion of U&SA materials.

C4 is an annotation on the diagram to document the responsibilities of a "good" plugin, that is, a plugin that supports the level of usability required by the MERBoard developers and its users. Since third parties often develop the plugin components, comprehensive documentation of any architectural decision to allocate responsibilities to a plugin must be provided so that these parties realize what conditions their code is expected to handle. This annotation came from a discussion of U&SA's Supporting Undo, Working at the User's Pace, and Observing System State scenario packages. Unlike the previous change, this change emerged from the discussion of several scenario packages rather that just one. This suggests that the combined effects of several scenario packages may produce considerations that do not arise when those scenario packages are considered singly.

The Plugin Services component (C5) was added in response to a discussion of the U&SA scenario Operating Consistently Across Views. This was the first scenario discussed to bring up the idea of having different views on the same data, therefore, it initiated a discussion of views themselves. The architecture expert connected this discussion to preliminary ideas about an object model presented at the first face-to-face meeting. He noted that there would be a lot of commonality between functions that manipulate aspects of the object model and proposed that common code inherited into the plugins would be better than making each plugin

implement these common functions themselves. FED agreed and added the Plugin Services component with the View Manager as an example.

Finally, the E-mail Manager (C6) was added shortly after the Plugin Services component as the discussion of multiple views continued. The MERBoard designers had envisioned that scientists may collaborate for a while using MERBoard, then analyze data in various ways on their own laptop or desktop computers, depending on individual interests. Thus, they expected that data on the MERBoard would have to be transferred to other computers. While discussing other aspects of multiple views, FED explained that this may be done through e-mail. As soon as he mentioned using e-mail, he noticed that he had not included a component representing the e-mail manager and added it. Although the e-mail function is not specifically tied to maintaining multiple views of data (the discussion underway), its omission was discovered as a direct consequence of discussing U&SA scenarios.

In summary, many changes were made to the proposed architecture to better support the usability goals of the MERBoard team. These changes included changing communication paths, adding components, and documenting aspects of the architecture not represented by lines and boxes. The first few changes were not linked to any specific U&SA scenario and might have been made during any architecture design walkthrough that included a usability expert (not usually present in current practice) and an architecture expert. However, when we examined each scenario in turn, we made some changes that specifically supported the scenario under consideration. Some changes related to only one scenario; others to a collection of scenarios that triggered a single solution. These cases clearly show that the U&SA materials influenced the final design of the MERBoard architecture.

7. SUMMARY OF FINDINGS

As we've shown above, the application of our U&SA materials to MERBoard's architecture enjoyed a measure of success. Now, we revisit the questions, expressed earlier, that we hoped a real-world application would be able to answer.

> *Would a real-world software development team accept the U&SA materials as the main discussion point of an architecture design meeting?*

We found that the entire MERBoard design and development team was not only willing to accept U&SA as the main discussion point, but actively

participated in a three hour review of their system based around our scenario packages. Moreover, we found that the discussion of our scenario packages included the participation of usability professionals who were silent during the conventional architecture presentation. This is encouraging, for it provides evidence that U&SA helps to improve communication between the software development and usability communities, one of its stated goals.

Would usability scenarios generated by considering single-user-at-a-desktop apply to a real-world design problem that may involve other domains (such as collaborative workspaces, web-based environments, etc)?

The MERBoard is a wall-sized collaborative workspace intended for use in a co-located, war-room style environment. It is a far cry from the single-user-at-a-desktop paradigm that we originally considered when developing the list of scenario packages, yet the MERBoard team still identified 25 of our 27 scenarios as applicable to their project; 17 of these being critical for the 2004 MER mission. Moreover, the team was able to give concrete examples of how the scenarios were realized for their users, often from their experiences performing direct observations of user behavior in the field trials.

We are encouraged to discover that so many scenarios were applicable in a CSCW application, since it implies that the scope of our technique lies beyond the single-user-at-a-desktop paradigm. Although we currently do not know how far our materials' range extends, this case provides evidence that they can be useful in at least one additional domain.

Would our architecture design suggestions contribute to a real design project?

As we have shown, the proposed architecture redesign for the MERBoard was heavily influenced by the front-end developer reading the U&SA documents and participating in an architectural review with the research team. The front-end developer felt that most of the materials were clear and relevant to his design. He especially liked the list of responsibilities the software must fulfill to support a usability scenario. During the detailed review, a majority of the architecture's components were modified to take into account the issues raised by U&SA scenario packages. U&SA clearly contributed to the architecture design of MERBoard.

However, we found that the architecture design patterns were less usable for the front-end developer than we had hoped and that he seemed to think the software tactics were irrelevant to his design. We have thus changed our approach with respect to these patterns, as discussed in the next section.

8. ONGOING WORK

Our ongoing work was influenced in several ways by our findings from applying U&SA materials to the MERBoard architecture redesign. In particular, we have redesigned our scenario packages and we are testing the efficacy of the different components of those packages in a more controlled setting.

We have found, both through our work with MERBoard and our experience teaching the U&SA materials, that the architecture design patterns we provide (Bass, John & Kates, 2001) as part of the scenario package are often insufficient for development purposes. Most developers find that our patterns are either not sufficiently general to be applicable to their system, or are so general that they have difficultly seeing how to apply them to their system. At the same time, both the MERBoard front-end developer and the participants in recent tutorials and classes find the architecture-independent lists of responsibilities that must be fulfilled to support a scenario extremely useful. This feedback led us to distinguish between architecture-independent responsibilities, architectural support for those responsibilities, and overarching architectural decisions related to aspects of the system other than usability. We have redesigned our scenario packages around this distinction, emphasizing responsibilities and rationale for the responsibilities (John, Bass, Sanchez-Segura, and Adams, 2004), in packages that are called *usability-supporting architectural patterns* (USAPs).

Encouraged that USAPs will be useful in software architecture design, we have collaborated with researchers on the European Union project called STATUS.[9] Some members of STATUS have also investigated the relationship between usability and software architecture (e.g., Bosch & Juristo, 2003; Folmer, and Bosch, 2004; Folmer, van Gurp, and Bosch, 2003; Juristo, Lopez, Moreno, and Sanchez-Segura, 2003). We expect that our combined effort will produce more USAPs than our research group could alone.

To investigate whether different pieces of the scenario packages contribute to the quality of a resulting architecture design, we are currently conducting a controlled laboratory experiment with software architects. The experiment compares three conditions: (1) only a scenario is given and the software architect is free to make architecture design changes as he or she sees fit, (2) giving both a scenario and the list of architecture-independent responsibilities to support that scenario, and (3) giving a scenario, the list of responsibilities, and a sample architecture pattern expressed in UML

[9] See the STATUS website http://www.ls.fi.upm.es/status/

component and sequence diagrams. Preliminary analyses show a significant improvement in the number of responsibilities considered by software designers when using responsibilities and UML diagrams over the scenario alone, and a trend toward improvement when using the list of responsibilities alone (Golden, John & Bass, 2005). We are continuing the analysis to assess quality of the architecture design. These data provide guidance to support future development of USAPs.

Finally, we realize that this chapter provides just part of the story about the usefulness of considering usability in architecture design. This chapter stops at an informal analysis of the creation of an architecture component diagram that supports the desired usability aspects of a system. However, there are many other questions to answer in the full development process. Did support for the scenarios get implemented at all? Was the architecture as designed sufficient to support the actual implementation of the scenarios or was it changed along the way? Did the end-users of MERBoard need the usability features supported by the architecture? Did they need even more support? We are currently analyzing many aspects of the development process, the implemented code, documentation, and actual user data during the MER 2004 mission to construct a more formal case study of this experience.

We also realize that a single case study cannot answer all questions regarding our materials. We are actively soliciting additional development groups wanting to explore their architecture designs from a usability viewpoint to gain more insight into the extent of U&SA materials' applicability and usefulness and to improve their design for the software architects who are our users.

9. ACKNOWLEDGEMENTS

The authors would like to thank the MERBoard development team for their willingness to participate in this research and for their insightful feedback on the U&SA materials. The Computational Sciences Division at NASA Ames Research Center provided support for MERBoard development and the MERBoard team's participation in our intervention. The High-Dependability Computing Program (HDCP) provided funding for the development of U&SA materials and through its testbed program provided us access to the MERBoard project for the purposes of testing the materials. Carnegie Mellon University's Software Engineering Institute (SEI) provided funding for the early development of U&SA via funding to Bonnie John as well as continuing support for the work of Len Bass.

10. REFERENCES

1. Bass, L., Clements, P., & Kazman, R. (1998). *Software Architecture in Practice, First Edition.* Reading, Massachusetts: Addison-Wesley.
2. Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice, Second Edition.* Reading, Massachusetts: Addison-Wesley.
3. Bass, L. & John, B. E. (2003) Linking usability to software architecture patterns through general scenarios. *Journal of Systems and Software, 66* (3), 187-197.
4. Bass, L. J, John, B. E. & Kates, J. (2001) *Achieving Usability Through Software Architecture*. Carnegie Mellon University/Software Engineering Institute Technical Report No. CMU/.SEI-2001-TR-005.
5. Bass, L., John, B. E., Juristo, N., Sanchez-Segura, M-I. (2004). *Usability and software architecture*. Tutorial materials presented at the 26th International Conference on Software Engineering, ICSE 2004 (Edinburgh, Scotland, May 23-28, 2004).
6. Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Boston, Massachusetts: Addison-Wesley.
7. Bosch, J. & Juristo, N. (2003) *Designing software architectures for usability*. Tutorial materials presented at the 25th International Conference on Software Engineering, ICSE 2003 (Portland, Oregon, May 3-10, 2003).
8. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. (1996). *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons.
9. Folmer, E. & Bosch, J. (2004) Architecting for usability. *Journal of systems and software (70),* 1., pp 61-78.
10. Folmer, E., van Gurp, J., & Bosch, J. (2003) Investigating the relationship between software architecture and usability. *Software Process - Improvement & Practice: Special Issue on Bridging the Process and Practice Gaps between Software Engineering and Human Computer Interaction*.
11. Fowler, M. (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition*. Reading, Massachusetts: Addison-Wesley.
12. Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). *Design Patterns*. Boston, Massachusetts: Addison-Wesley.
13. Golden, E., John, B. E., & Bass, L. (2005), The value of a usability-supporting architectural pattern in software architecture design: A controlled experiment. Proceedings of the Intrenational Conference on Software Engineering, ICSE 2005, (St. Louis, Missouri, 15-21 May, 2005
14. ISO 9241-11:1998, *Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability*.
15. John, B. E., Bass, L., Juristo, N., Sanchez-Segura, M-I. (2004) Avoiding *"We can't change THAT!": Software Architecture and Usability.* Tutorial materials presented at CHI 2004 (Vienna, Austria, April 24-29, 2004).
16. John, B. E. & Bass, L. (2003) Avoiding *"We can't change THAT!": Software Architecture and Usability.* Tutorial materials presented at CHI 2003 (Ft. Lauderdale, FL, April 5-10, 2003).
17. John, B. E. & Bass, L. (2002) Avoiding *"We can't change THAT!": Software Architecture and Usability.* Tutorial materials presented at CHI 2002 (Minneapolis, MN, April 5-10, 2002).
18. John, B. E., Bass, L. J., Sanchez-Segura, M-I. & Adams, R. J. (2004) Bringing usability concerns to the design of software architecture. *Proceedings of The 9th*

IFIP Working Conference on Engineering for Human-Computer Interaction and the 11th International Workshop on Design, Specification and Verification of Interactive Systems, (Hamburg, Germany, July 11-13, 2004).

19. Juristo, N, Lopez, M., Moreno, A. M., & Sanchez-Segura, M-I. (2003) Improving software usability through architectural patterns. In *Proceedings of the ICSE 2003 Workshop Bridging the Gaps Between Software Engineering and Human-Computer Interaction*. Portland (Oregon), USA, May 2003. pp. 12-19.
20. Kawakita, J: *The original KJ-method*. Kawakita Research Institute, Tokyo 1982.
21. Kazman, R., Klein, M., Clements, P. (2000). *ATAM: Method for Architecture Evaluation.* CMU/SEI-2000-TR-004. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.
22. NASA Ames Research Center (2003). *MERBoard User's Guide*. Moffet Field, CA.
23. Newman, W. & Lamming, M. (1995). *Interactive System Design*. Wokingham, England: Addison-Wesley Publishing.
24. Nielsen, J. (1993). *Usability Engineering*. Boston, MA: Academic Press Inc.
25. Norman, D. A., (1983) Design rules based on analyses of human error. *Communications of the ACM*. (New York: ACM Press) pp. 254-258.
26. Rosson, M. B., & Carroll, J. M. (1992) Getting around the task-artifact cycle: how to make claims and design by scenario. *ACM Transactions on Information Systems 10*, (2) pp. 181 – 212.
27. Shneiderman, B. (1998). Designing the User Interface, Third Edition. Reading, MA: Addison-Wesley.
28. Sun Microsystems, Inc. (2003). Model-View Controller. Java Blueprints. Retrieved September 18th, 2003 from the World Wide Web: http://java.sun.com/blueprints/patterns/MVC-detailed.html
29. Tollinger, I. McCurdy, M., Vera, A., & Tollinger, P. (2004) Collaborative knowledge management supporting Mars mission scientists. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW 2004* (Chicago, Nov. 6-10, 2004).
30. Wirfs-Brock, R. & Mckean, A, *Object Design: Roles, Responsibilities, and Collaborations*. Addison Wesley, Reading, Ma, 2003.